Avignon Université - Culture numérique et code

CM - Fonctions

06/03/2020

Définir une fonction

L'approche efficace d'un problème complexe consiste souvent à le décomposer en plusieurs sous-problèmes plus simples qui seront étudiés séparément.

D'autre part, il arrivera souvent qu'une même séquence d'instructions doive être utilisé à plusieurs reprises dans un programme, et on souhaitera bien évidemment ne pas avoir à la reproduire systématiquement.

Définition de la fonction :

```
def nomDeLaFonction(liste de praramètres):
    ...
    #bloc d'instructions
    return valeur
    ...
```

Appel à fonction :

```
...
reponse = nomDeLaFonction(liste de praramètres)
print(reponse)
...
```

Exemple :

In [1]:

```
def addition(num1, num2):

    print("Premier opérande :", num1)
    print("Deuxième opérande :", num2)

    resultat = num1 + num2
    return resultat

res1 = addition(10, 3)
print(res1)

res2 = addition(5.5, 12.5)
print(res2)
```

```
Premier opérande : 10
Deuxième opérande : 3
13
Premier opérande : 5.5
Deuxième opérande : 12.5
18.0
```

> **Exercice 1 :** Ajouter une fonction `soustraction` à l'exemple antérieur. La fonction doit recevoir deux paramètres et retourner la soustraction du deuxième paramètre moins le premier. Tester la fonction.

Variables locales et globales

Lorsque nous définissons des variables à l'interieur du corps d'une fonction, ces variables ne sont accessibles qu'à la fonction elle-même. On dit que ces variables sont des **variables locales** à la fonction. C'est par exemple le cas des variables `num1` , `num2` et `resultat` de l'exemple suivant.

```
def addition(num1, num2):

    print("Premier opérande :", num1)
    print("Deuxième opérande :", num2)

    resultat = num1 + num2
    return resultat

addition(10,3)
print(resultat)
```

Ainsi l'instruction `print(resultat)` produira l'erreur suivant : `NameError: name 'resultat' is not defined` .

Les variables définies á l'extérieur d'une fonction sont des **variables globales**. Leur contenu est "visible" de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier.

Exemple :

In [2]:

```
a = 1

def fonction1():
    print("fonction1 :", a)
    return

def fonction2():
    a = 2
    print("fonction2 :", a)
    return

def fonction3(a):
    print("fonction3 :", a)
    a = 3
    print("fonction3 :", a)
    return


fonction1()
fonction2()
fonction1()
fonction3(10)
fonction1()
```

```
fonction1 : 1
fonction2 : 2
fonction1 : 1
fonction3 : 10
fonction3 : 3
fonction1 : 1
```

Instruction `global`

Cette instruction permet d'indiquer (à l'intérieur de la définition d'une fonction) quelles sont les variables à traiter globalement.

Exemple :

In [3]:

```
a = 1

def fonction1():
    print("fonction1 :", a)
    return

def fonction2():
    global a
    a = 2
    print("fonction2 :", a)
    return

def fonction3(a):
    print("fonction3 :", a)
    a = 3
    print("fonction3 :", a)
    return


fonction1()
fonction2()
fonction1()
fonction3(10)
fonction1()
```

```
fonction1 : 1
fonction2 : 2
fonction1 : 2
fonction3 : 10
fonction3 : 3
fonction1 : 2
```

> **Exercice 2 :** Copier le script `operations1.py` , l'exécuter et repondre aux questions suivantes :

1. Combien de fonctions il y a ?
2. Pour quoi dans la fonction `demander_nom` est-il nécessaire l'instruction `global` pour la variable `nom` ?
3. Pour quoi dans la fonction `main` il n'est pas nécessaire l'instruction `global` pour la variable `nom` ?
4. Les variables `num1` , `num2` et `resultat` de la fonction `addition` sont elles les mêmes que ceux de la fonction `soustraction` ?
5. Est-il possible d'ajouter le code suivate dans la fonction `main` ? Dans ce cas, quel est le résultat ?

```
print("Résultat:", soustraction(multiplication(addition(3, 2), 4), 5))
```

6. Ajouter une fonction `cube` qui reenvoie le cube d'un nombre et tester.

```python
# operations1.py
# Auteur : Carlos González
# Date : 3/02/2020
# Le programme demande deux valeurs et compute des opérations.

nom = ""

def addition(num1, num2):
    resultat = num1 + num2
    return resultat

def soustraction(num1, num2):
    resultat = num1 - num2
    return resultat

def multiplication(n1 , n2):
    print("Dans la fonction 'multiplication'...")
    return n1 * n2

def demander_nom():
    global nom
    nom = input("Donne moi ton nom : ")
    return

def demander():
    valeur = float(input("Donne moi une valeur : "))
    return valeur

def main():

    demander_nom()
    print("Bonjour", nom, "!")

    val1 = demander()
    val2 = demander()

    res1 = addition(val1, val2)
    print("Addition :", res1)

    print("Multiplication :", multiplication(val1, val2))

    res2 = soustraction(val2, val1)
    print("Soustraction :", res2)


main()
```

Modules de fonctions

Il existe un grand nombre de modules pré-programmés qui sont fournis d'office avec Python. Souvent on essaie de regrouper dans un même module des ensembles de fonctions apparentées, que l'on appelle des *bibliothèques*.

Le module *math*, contient les définitions de nombreuses fonctions mathématiques et variables. Pour pouvoir utiliser ses fonctions et variables, il suffit d'incorporer la ligne suivante au début du script.

```
from math import *
```

Bibliothèque math : https://docs.python.org/3/library/math.html (https://docs.python.org/3/library/math.html)

Exemple :

In [4]:

```
from math import *

def main():

    num = 9
    print("La valeur de pi est :", pi)
    print("La racine carrée de", num, "est :",sqrt(num))
    print(num, "puissance 2 :", pow(num, 2))
    print(num, "puissance 3 :", pow(num, 3))
main()
```

```
La valeur de pi est : 3.141592653589793
La racine carrée de 9 est : 3.0
9 puissance 2 : 81.0
9 puissance 3 : 729.0
```

> **Exercice 3 :** Créer un script séparé en trois fonctions (`main` , `cube` , `volumeSphere`) qui calcule le volume d'une sphere. Le script doit utiliser la bilbiothèque *math*.

TP - Fonctions

Exercice 1

Écrire un script qui obtient la moyenne, médiane, min, max, variance et l'écart type d'un liste de valeurs.

In [ ]:

```

```